Class Overview

Large Language Models: Methods and Applications

Daphne Ippolito and Chenyan Xiong

cmu-llms.org

Instructors

Chenyan Xiong
Office hours: Thursdays 3:30-4
GHC 6409

Chenyan Xiong
Office hours: Tuesdays 3:45-4:15
GHC 6407

Plus guest lectures industry experts!

Teaching Assistants

Cathy Jiao

Joao Coelho

Ava Yan

Kshitish Ghate

Xinyue Liu

Nishant Subramani

Yiming Zhang

Harshita Diddee

How to reach us

Questions about the lecture or homework material:
Piazza or office hours.

General logistics questions:
Piazza *or* office hours *or* email to llms-11-667@andrew.cmu.edu.

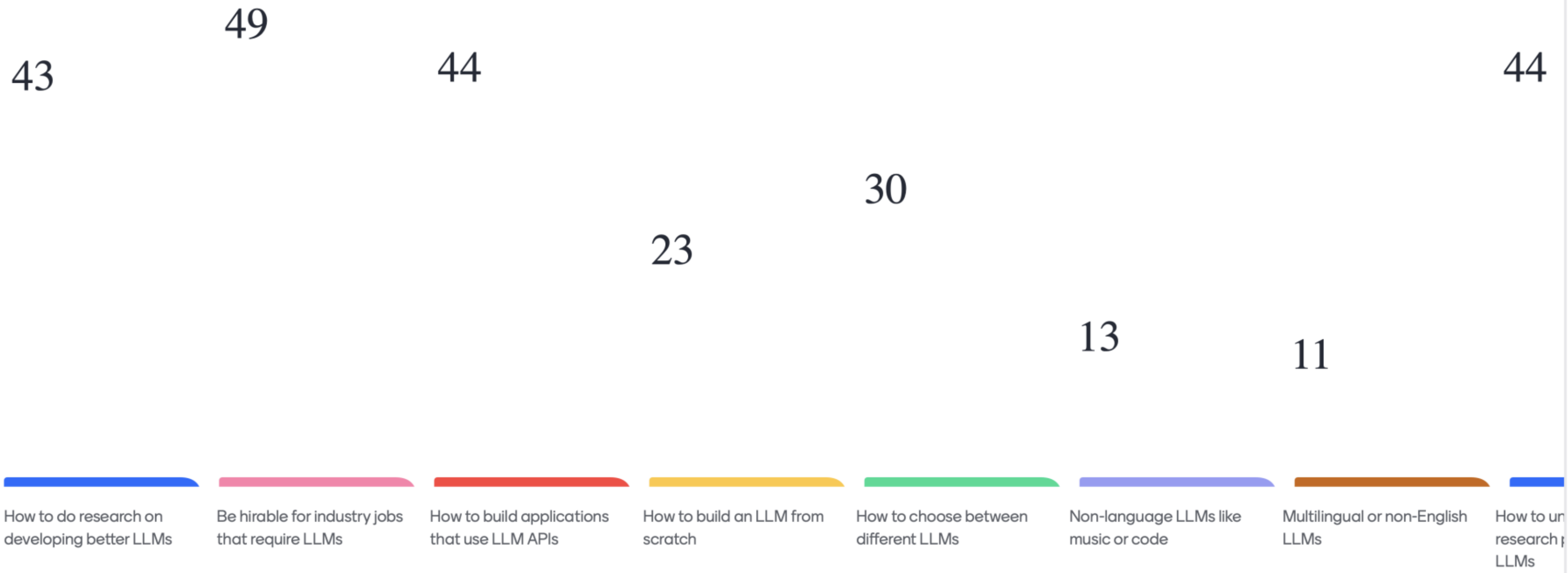
Questions about specific situations (missing class, grades, etc.):
Professor office hours *or* private Piazza post *or* email to llms-11-667@andrew.cmu.edu.

Emails sent to individual TAs or instructors will be ignored.

SURVEY

Join at menti.com | use code 2316 9150

What are hoping to learn from this class?



Account

Content

Design

Settings

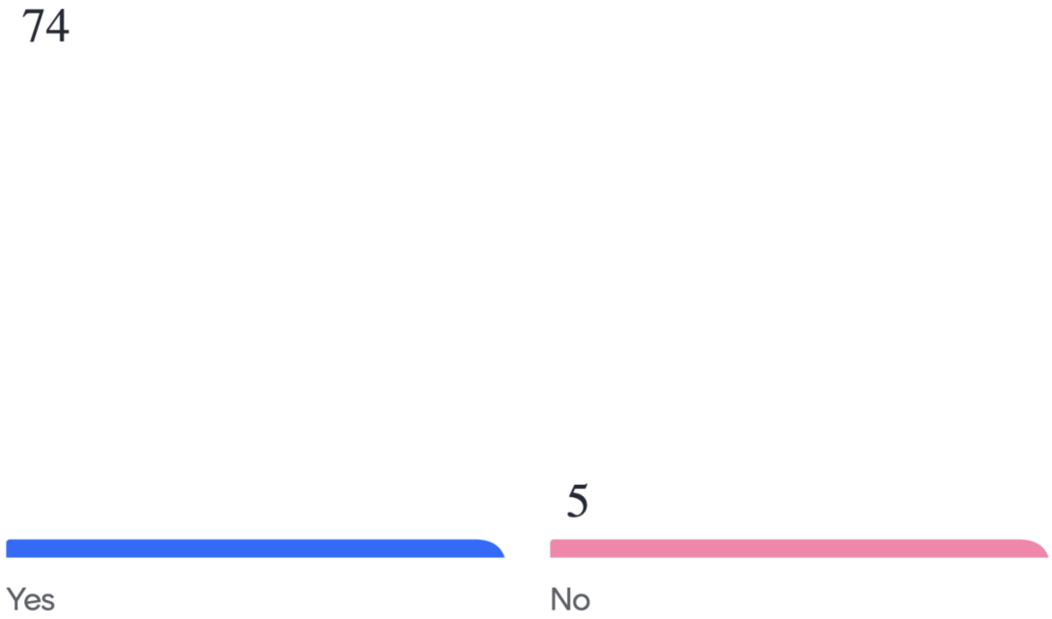
Help & Feedback

Join at menti.com | use code 2316 9150

Share other reasons you are taking this class.

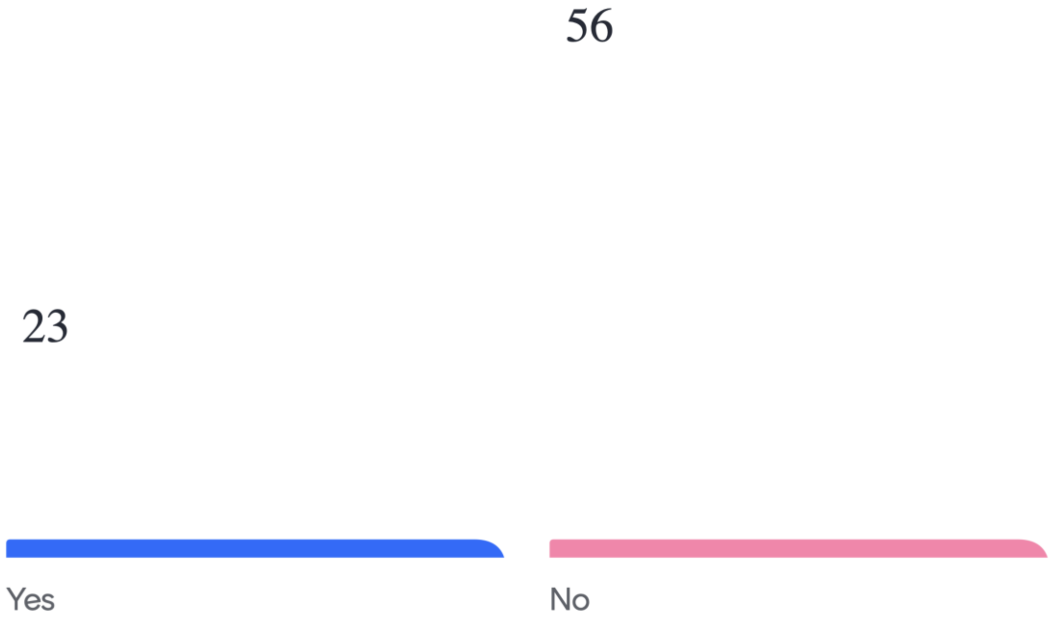
Join at menti.com | use code 2316 9150

Have you taken a machine learning or deep learning class before?



Join at menti.com | use code 2316 9150

Have you taken an NLP or computational linguistics class before?



Account
Content
Design
Settings
Help & Feedback

Join at menti.com | use code 2316 9150

How familiar are you with?

Python
6.6
PyTorch
5.6
HuggingFace
4.0
ChatGPT
6.5
Bard
2.7
Copilot
4.3
OpenAI API
4.1
Other LLM APIs
3.1

Never heard of it

Have used it extensiv

Account

Content

Design

Settings

Help & Feedback

After successfully completing this task, you should be able to…

- Compare and contrast different models in the LLM ecosystem in order to determine the best model for any given task.
- Implement and train a Transformer-based language model from scratch in Pytorch.
- Utilize open-source libraries to finetune and do inference with popular pre-trained language models.
- Understand how to apply LLMs to a variety of downstream applications, and how decisions made during pre-training affect suitability for these tasks.
- Read and comprehend recent, academic papers on LLMs and have knowledge of the common terms used in them (alignment, scaling laws, RLHF, prompt engineering, instruction tuning, etc.).
- Design new methodologies to leverage existing large scale language models in novel ways.

Assessment

- Six homework assignments (60%)
  - To be completed Individually
  - Mixture of practical and comprehension-based questions
  - To be turned in via Gradescope
- One midterm (20%)
- One final exam (20%)

Homeworks

Homework 1:
- Implement a Transformer from scratch
  Implement a tokenizer
- Inference with the HuggingFace API

Homework 2:
- Understand pre-training data curation decisions
- Implement a pre-training data pipeline

Homework 3:
- Retrieval-augmented generation
- Tool-use

Homework 4:
- How to choose between models
- Measuring and reducing bias

Homework 5:
- Improving training efficiency

Homework 6 (mini-project):
- Apply techniques learned in class to a task of your choice

What we expect from students taking this class:

- Fluent in Python
- Comfortable with with a Python deep learning framework such as PyTorch
- Able to commit ~9 hours per week to homework
- Have already taken courses in NLP and ML, or wiling to put in extra time to self-learn needed concepts as they come up

Waitlist Policy

- This is a popular class, but we expect most of you who are on the waitlist, if you stick around, will eventually get in.
- Come to class. Start on Homework 1.
- Let us know if you have done all but still cannot get in near Sep. 10th.

Language Model Basics

Large Language Models: Methods and Applications

Daphne Ippolito

Carnegie Mellon University

Agenda

1. What is a Language Model?
2. Building Blocks of Language Models
3. Decoding Strategies
4. Language Model Architectures

1. What is a Language Model?

What is a Language Model?

A language model is any model that outputs a probability distribution over the next token* in a sequence given the previous tokens in the sequence, that is: $P(y_t|y_{1:t-1})$.

Historically, language models were statistical n-gram models. Instead of taking into account the full history of the sequence, they approximated this history by just looking back a few words.

*For now, let's assume token = word. We'll come back this.

What is a Language Model?

Example: Suppose we are building a statistical language model using a text corpus, *C*. We observe that the word “apple” follows the words “eat the” 2% of the times that “eat the” occurs in *C*.

This means we’d set:

P(“apple” | “eat the”) = 0.02.

Since “eat the apple” is three words, we’d call this a 3-gram model.

Language models are not inherently generative.

Computing Sequence Likelihood

Language models output the likelihood of the next word: $P(y_t|y_{1:t-1})$.

Often we will talk about the likelihood of an entire sequence $P(Y) = P(y_1, y_1, \ldots, y_T)$.

Computing Sequence Likelihood

Sequence likelihood can be computed from an LM using the chain rule:

P([“I”, “eat”, “the”, “apple”]) =

P(“apple” | [“I”, “eat”, “the”]) * P(“the” | [“I”, “eat”]) * P(“eat” | [“I”]) * P(“I”])

In math:

$$P(Y) = P(y_1, y_2, \dots, y_T) = P(y_T | y_{1:T-1}) \times P(y_{T-1} | y_{1:T-2}) \times \cdots \times P(y_1 | \text{start of sequence})$$

Neural Language Models: Conditioned v. Unconditioned

Neural language models can either be designed to just predict the next word given the previous ones, or they can be designed to predict the next word given the previous ones *and* some additional conditioning sequence.

Unconditioned: $P(Y)$

At each step the LM predicts:

$P(y_t \mid y_{1:t-1})$

Examples:

- GPT-2 / GPT-3
- LLaMA

Conditioned: $P(Y \mid X)$

At each step the LM predicts: $P(y_t \mid y_{1:t-1}, x_{1:T})$

Examples

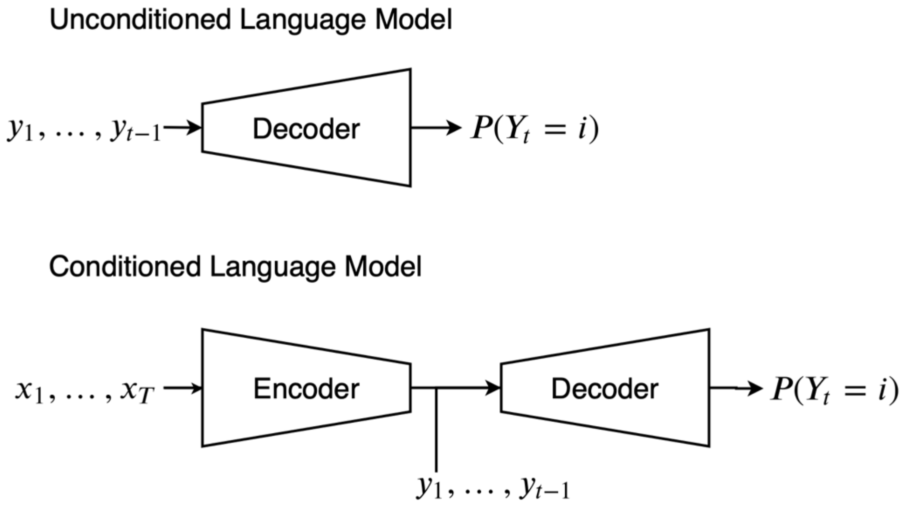
- T5
- Most machine translation models

Sometimes called sequence-to-sequence or seq2seq models.

2. Building Blocks of Language Models

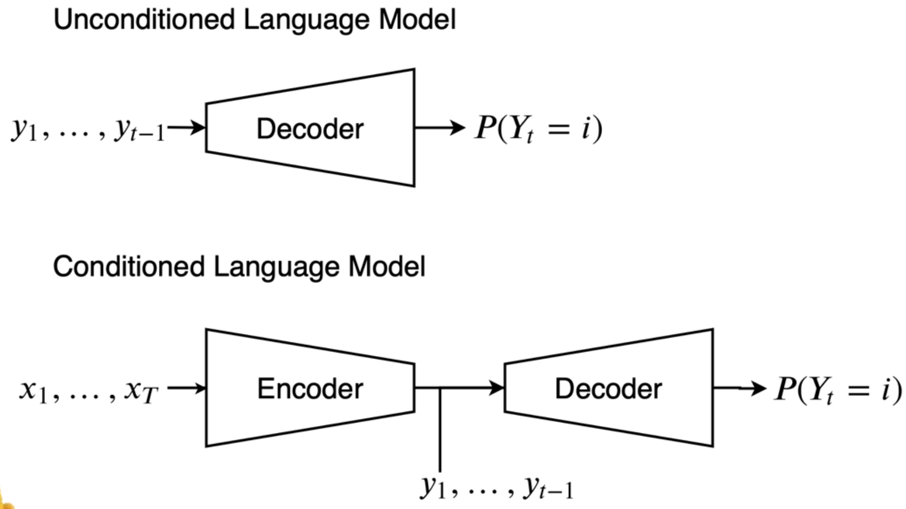
Neural Language Models: Conditioned v. Unconditioned

Unconditioned neural language models only have a decoder.
Conditioned ones have an encoder and a decoder.



Neural Language Models: Conditioned v. Unconditioned

Unconditioned neural language models only have a decoder. Conditioned ones have an encoder and a decoder.



There are also encoder-only models, but they aren't traditional language models.

Neural Language Models: Conditioned v. Unconditioned

Theoretically, any task designed for a decoder-only architecture can be turned into one for an encoder-decoder architecture, and vice-versa.

TASK: Continue the sequence.

Decoder-only version:

P(Y="Once upon a time there lived a dreadful ogre.")

Encoder-decoder version:

P(Y="lived a dreadful ogre." | X="Once upon a time there")

Neural Language Models: Conditioned v. Unconditioned

Theoretically, any task designed for a decoder-only architecture can be turned into one for an encoder-decoder architecture, and vice-versa.

TASK: Translate from English to French.

Decoder-only version:

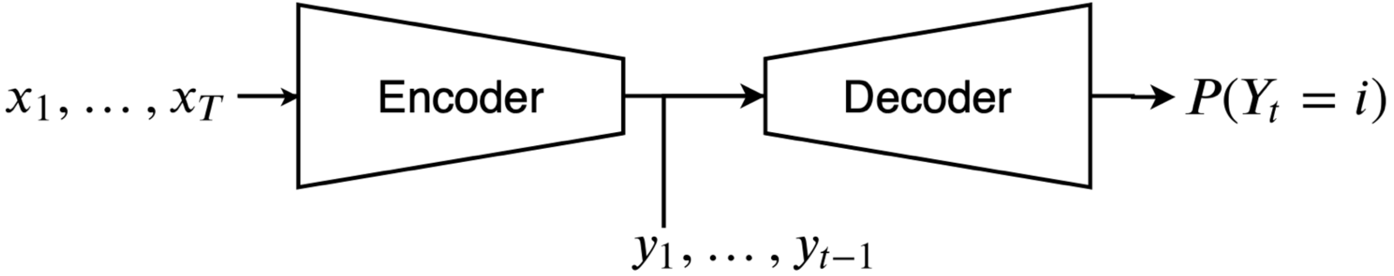
P(Y="English: The hippo ate my homework. French: L'hippopotame a mangé mes devoirs.")

Encoder-decoder version:

P(Y="L'hippopotame a mangé mes devoirs." | X="The hippo ate my homework.")

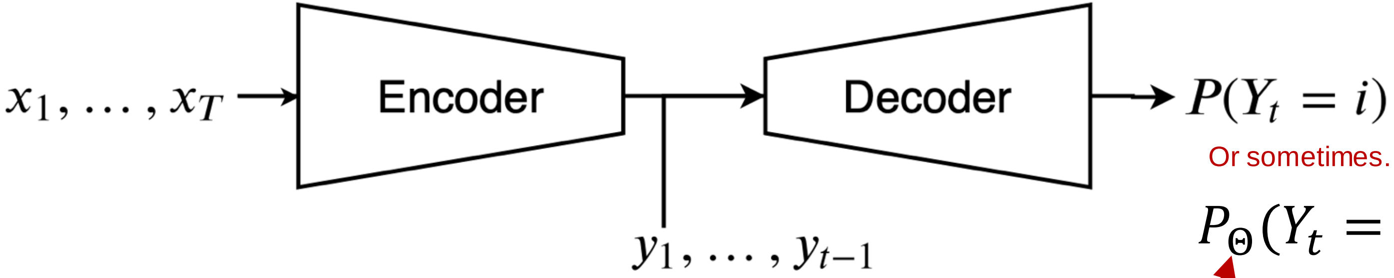
Summary of Terms You Should Know

Input sequence: $x_1, \ldots, x_T$
Target sequence: $y_1, \ldots, y_T$



Summary of Terms You Should Know

Input sequence: $x_1, \ldots, x_T$
Target sequence: $y_1, \ldots, y_T$



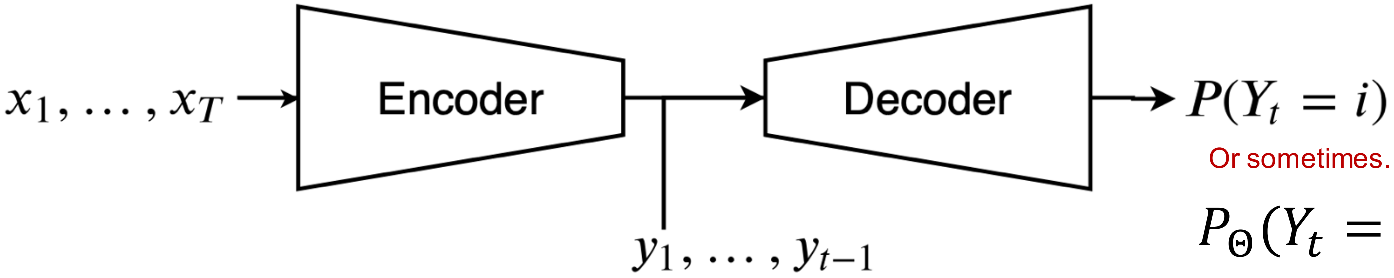
Or sometimes…

$$P_\Theta(Y_t = i)$$

Represents the parameters of the neural network.

Summary of Terms You Should Know

Input sequence: $x_1, \dots, x_T$
Target sequence: $y_1, \dots, y_T$



Or sometimes…

$$P_\Theta(Y_t = i)$$

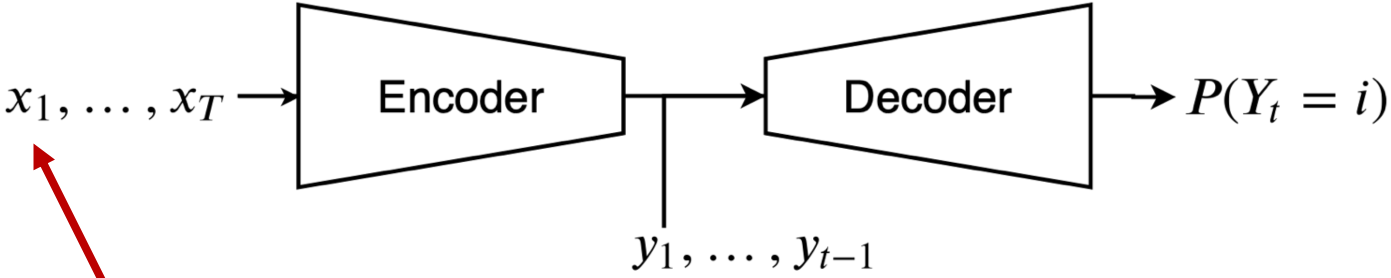
Or sometimes…

$$P(Y_t = i | y_1, \dots, y_{t-1}; x_1, \dots, x_T; \Theta)$$

Summary of Terms

Input sequence: $x_1, \ldots, x_T$

Target sequence: $y_1, \ldots, y_T$



What are x_i and y_i?

Tokenizing Text

Tokenization is the task of taking text (or code or music) and turning it into a sequence of discrete items, called tokens.

Tokenizing Text

A **tokenizer** takes text and turns it into a sequence of discrete **tokens.**

A **vocabulary** is the list of all available tokens.

Let's tokenize: "A hippopotamus ate my homework."

Vocab Type	Example	Ex. length
character-level	['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's', ' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']	31
subword-level	['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']	9
word-level	['A', 'hippopotamus', 'ate', 'my', 'homework']	5

Tokenizing Text

A **tokenizer** takes text and turns it into a sequence of discrete **tokens.**

A **vocabulary** is the list of all available tokens.

Let's tokenize: "A hippopotamus ate my homework."

Vocab Type	Example	Ex. length
character-level	['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's', ' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']	31
subword-level	['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']	9
word-level	['A', 'hippopotamus', 'ate', 'my', 'homework']	5

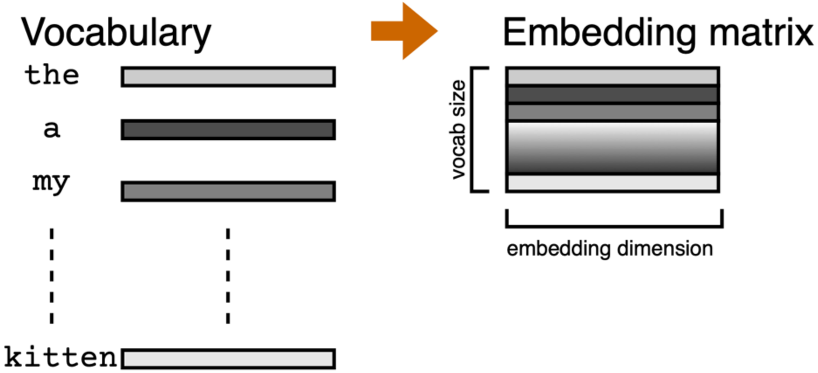
What are the pros and cons of different tokenizers?

More on this in two lectures!

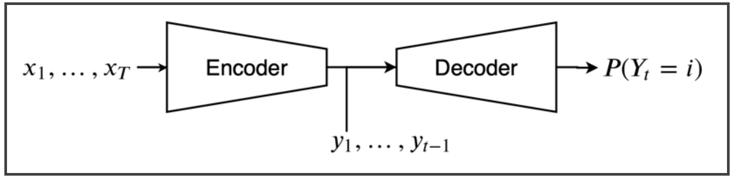
Turning Discrete Tokens into Continuous Vectors

Neural networks cannot operate on discrete tokens.

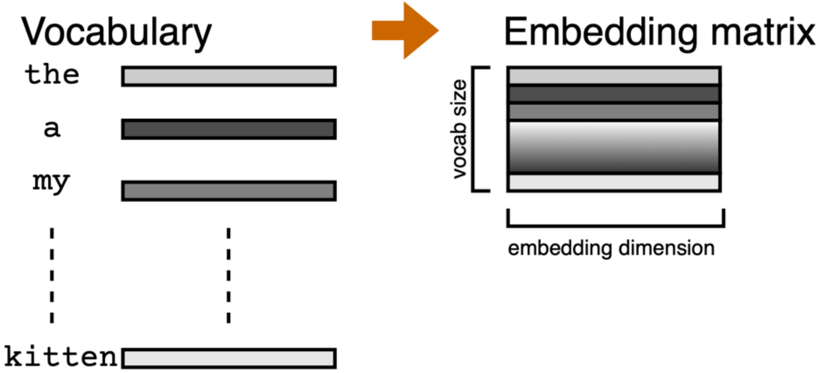
Instead, we build an **embedding matrix** which associates each token in the vocabulary with a vector embedding.

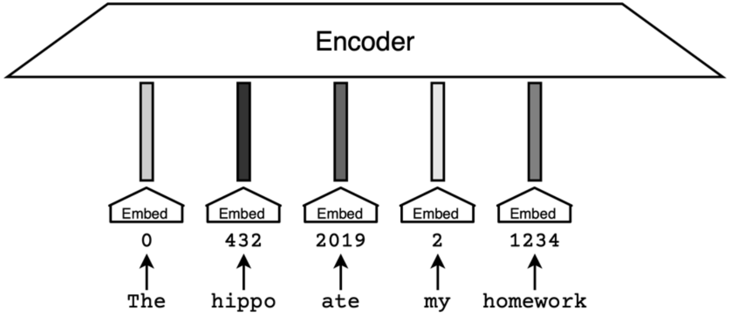


Encoder Inputs and Outputs

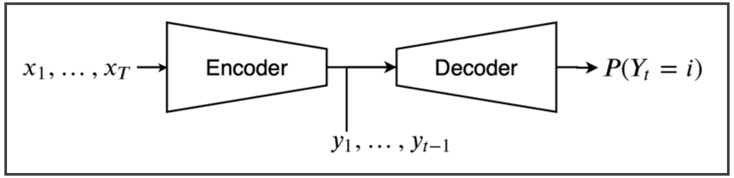


The encoder takes as input the vector representations of each token in the input sequence.

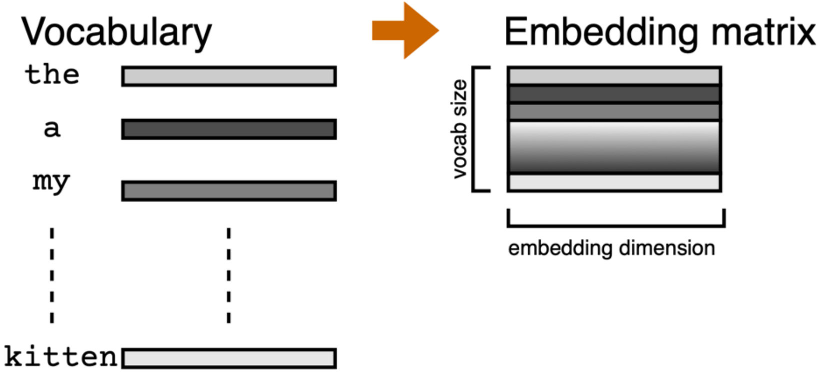


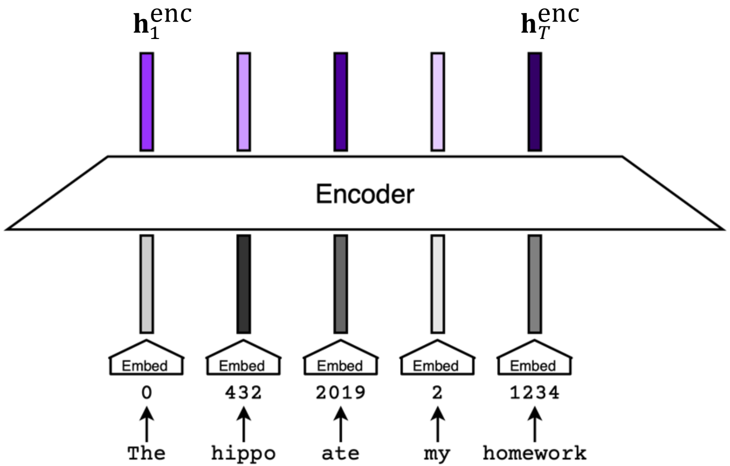


Encoder Inputs and Outputs

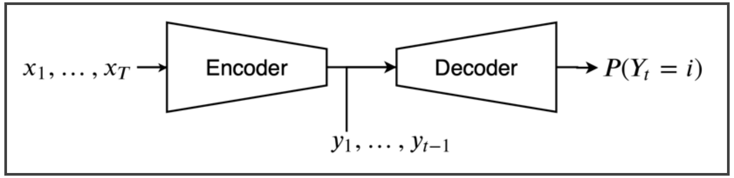


The encoder outputs a sequence of embeddings called **hidden states**.



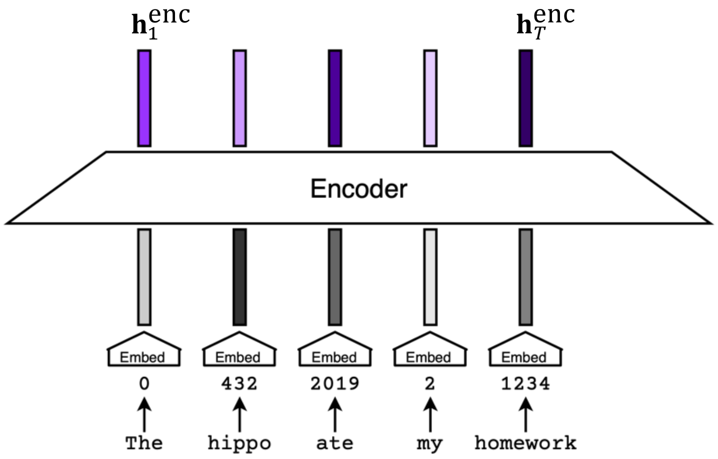


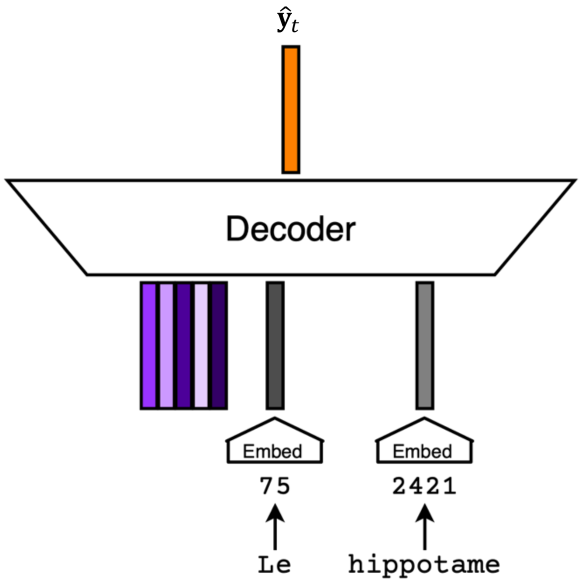
Decoder Inputs and Outputs



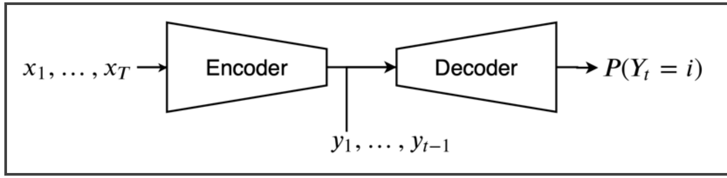
The decoder takes as input the hidden states from the encoder as well as the embeddings for the tokens seen so far in the target sequence.

It outputs an embedding $\hat{\mathbf{y}}_t$.

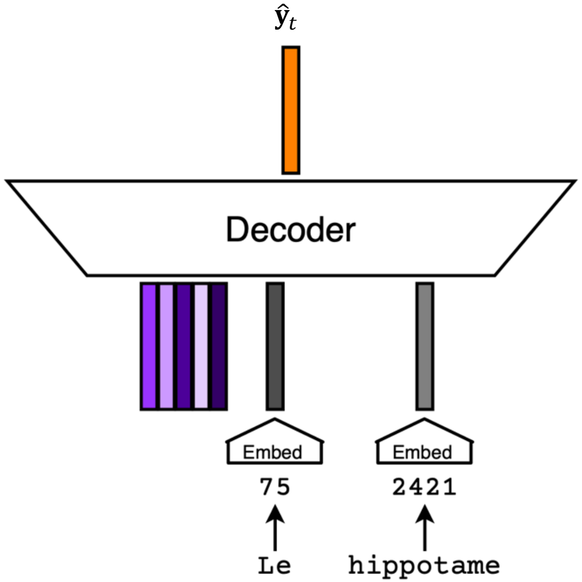




Decoder Inputs and Outputs

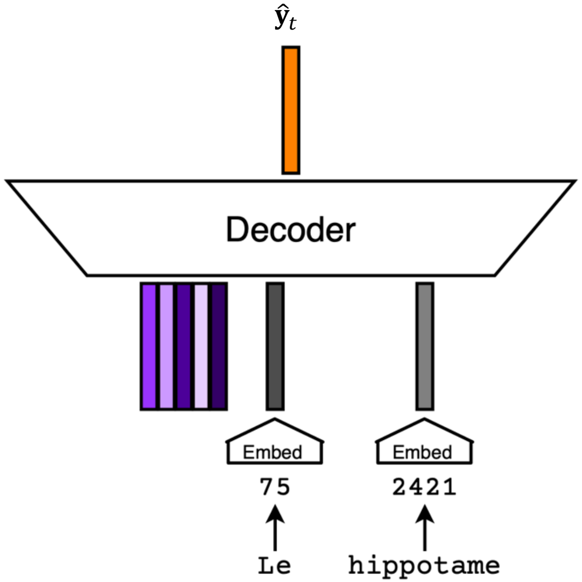


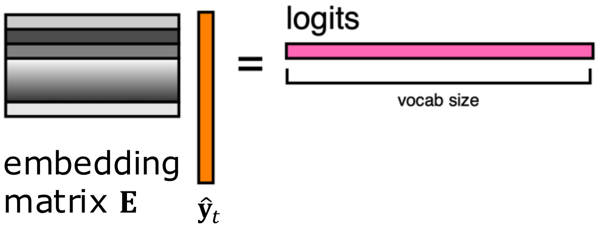
Ideally, $\hat{\mathbf{y}}_t$ would be as close as possible to the embedding of the true next token.



Decoder Inputs and Outputs

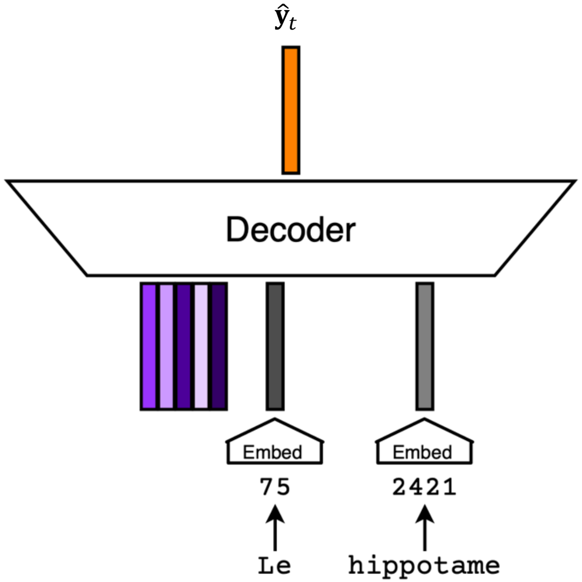
We multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.

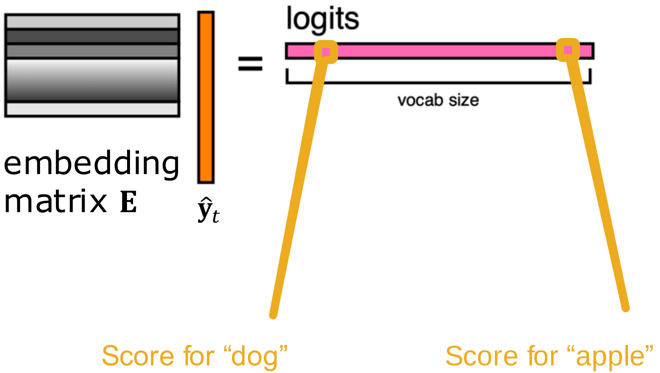




Decoder Inputs and Outputs

We multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.





Decoder Inputs and Outputs

We multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.

The softmax function is used to turn the logits into probabilities.

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Decoder Inputs and Outputs

We multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.

The softmax function is used to turn the logits into probabilities.

$$P(Y_t = i|\mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Example: Suppose we are trying to predict the 5th word in the sequence "the dog chased the". We want to know the probability the next word is "cat".

$$P(Y_5 = \text{"}cat\text{"}|\text{"}the\ dog\ chase\ the\text{"}) = \frac{\exp(\text{score in logits for "}cat\text{"})}{\text{normalization term}} = 0.321$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log \boxed{P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})}$$

The probability the language model assigns to the true t^{th} word in the target sequence.

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = \boxed{i^*} | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The index of the true t^{th} word in the target sequence.

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

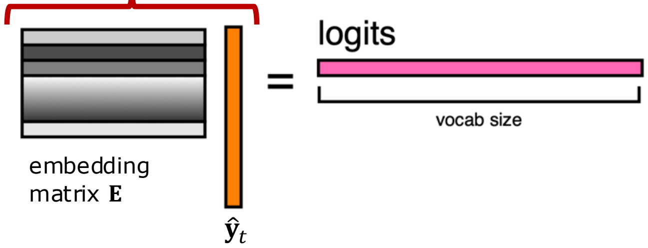
Recall:

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\boxed{\mathbf{E}\hat{\mathbf{y}}_t}[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$



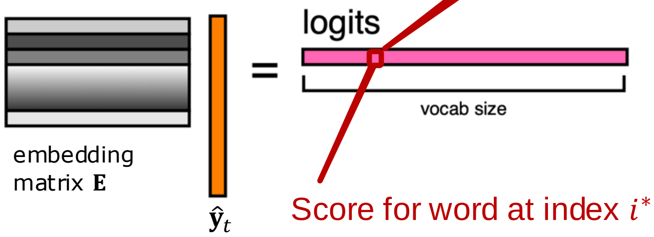
Recall:

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\boxed{\mathbf{E}\hat{\mathbf{y}}_t[i^*]})}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$



Recall:

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

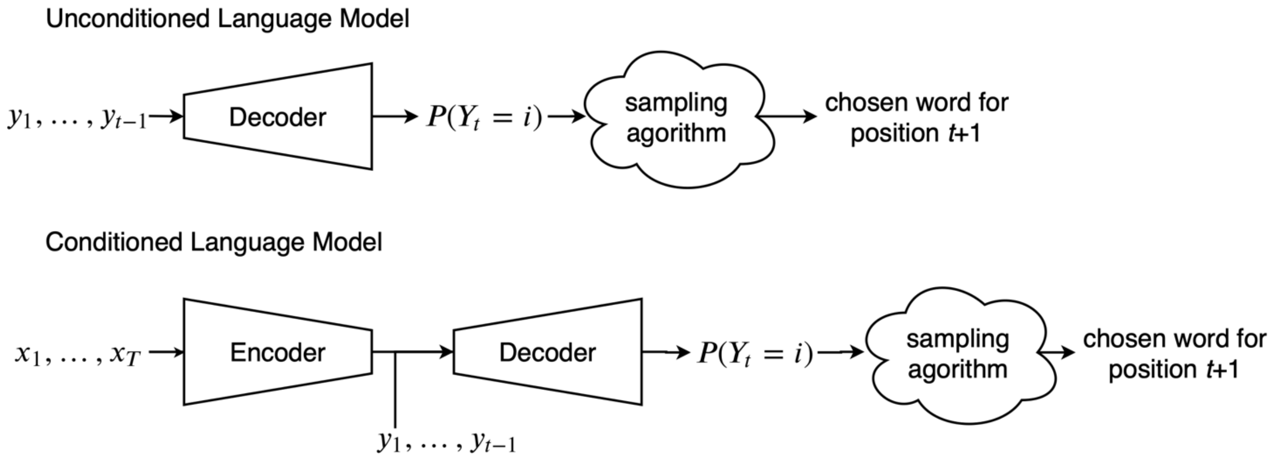
$$= -\sum_{t=1}^{T} \mathbf{E}\hat{\mathbf{y}}_t[i^*]$$

Recall:

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

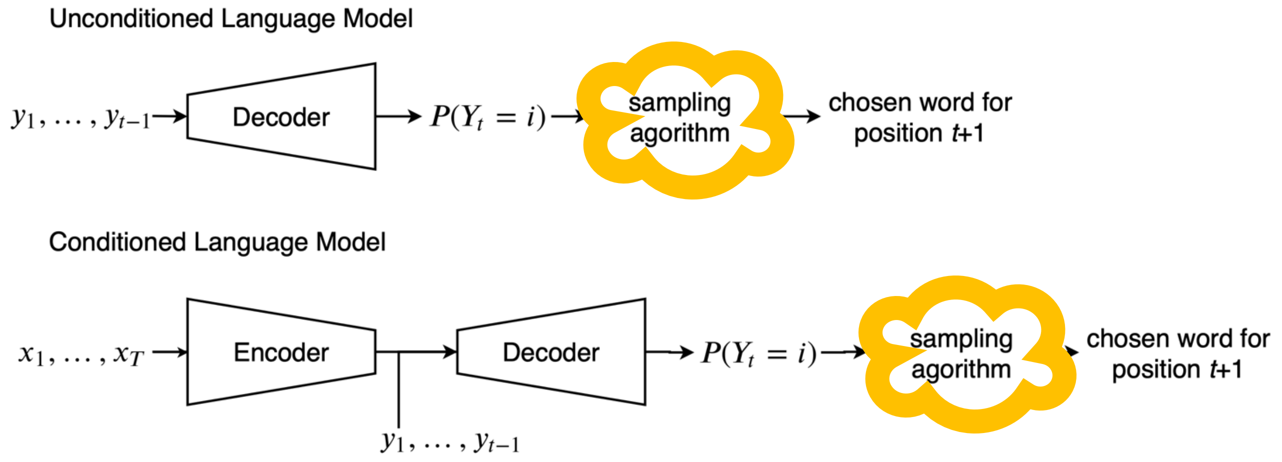
Now how do we do generation?

To do generation, we need a **sampling algorithm** that selects a word given the predicted probability distribution $P(Y_t = i|y_{1:t-1})$.



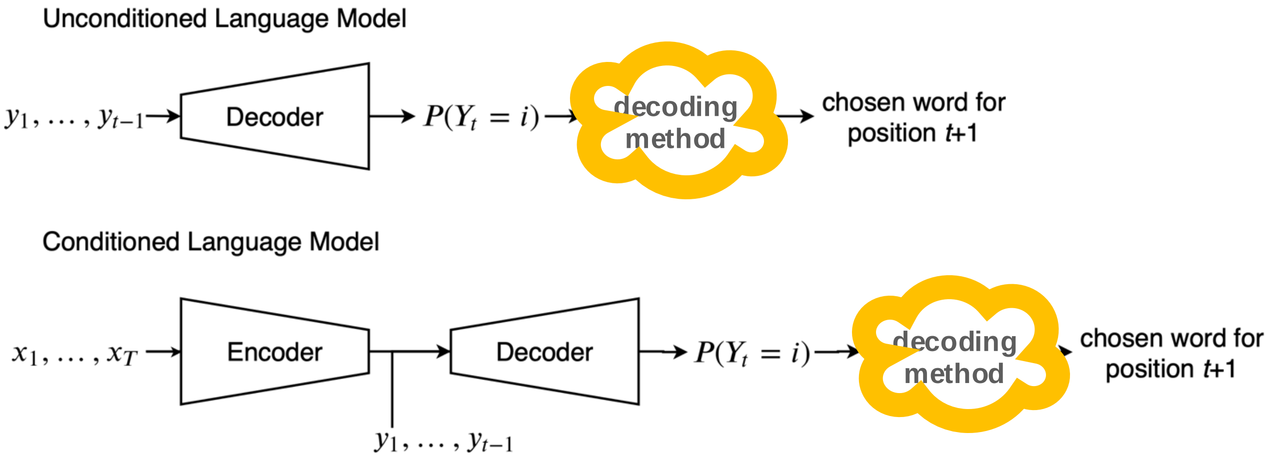
Now how do we do generation?

To do generation, we need a **sampling algorithm** that selects a word given the predicted probability distribution $P(Y_t = i|y_{1:t-1})$.



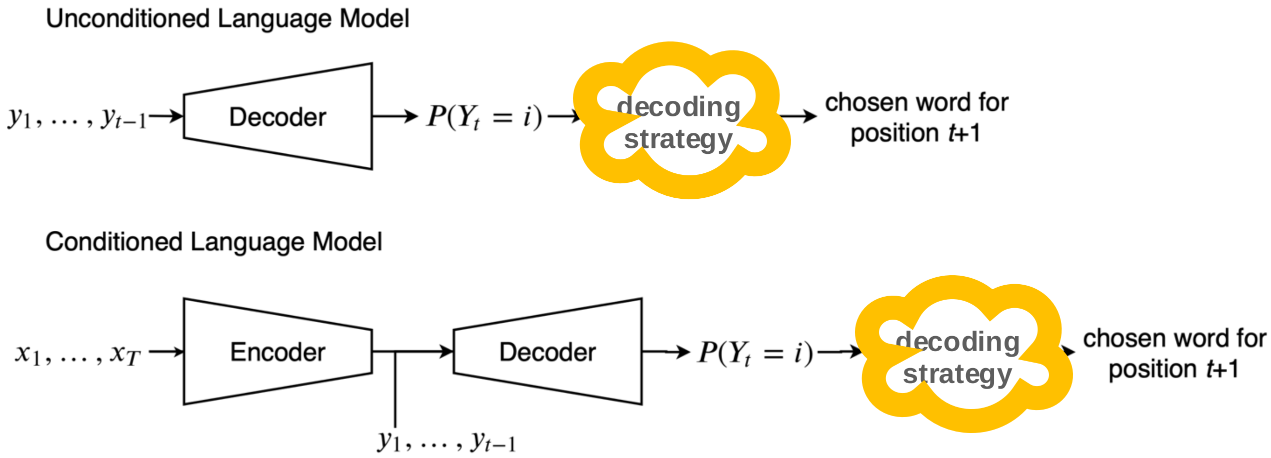
Now how do we do generation?

To do generation, we need a **sampling algorithm** that selects a word given the predicted probability distribution $P(Y_t = i|y_{1:t-1})$.



Now how do we do generation?

To do generation, we need a **sampling algorithm** that selects a word given the predicted probability distribution $P(Y_t = i|y_{1:t-1})$.



Questions so far?

3. Decoding Strategies

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

TYPE YOUR ANSWER INTO CHAT

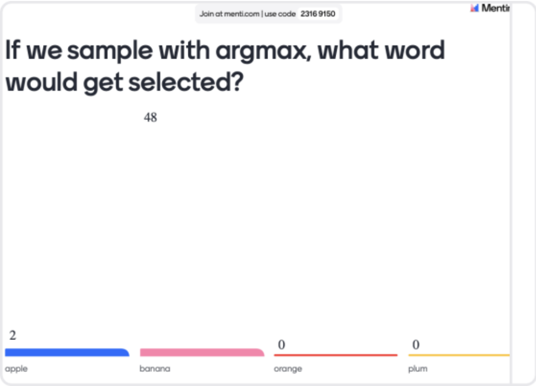
Suppose our vocab consists of 4 words:
$\mathcal{V} = \{apple, banana, orange, plum\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} | Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} | Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

If we sample with argmax, what word would get selected?

(a) apple (b) banana (c) orange (d) plum



How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{apple, banana, orange, plum\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

If we sample with argmax, what word would get selected?

(a) apple **(b) banana** **(c) orange** **(d) plum**

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

TYPE YOUR ANSWER INTO CHAT

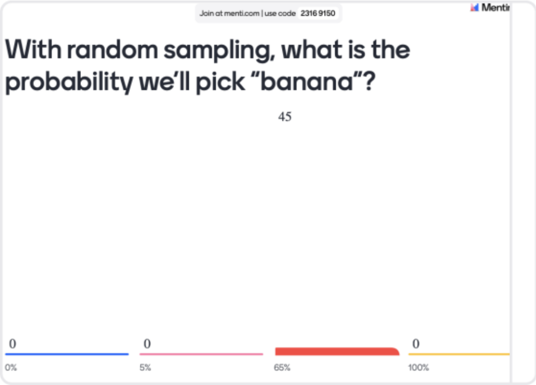
Suppose our vocab consists of 4 words:
$\mathcal{V} = \{apple, banana, orange, plum\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

With random sampling, what is the probability we'll pick "banana"?

(a) 0%   (b) 5%   (c) 65%   (d) 100%



How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{apple, banana, orange, plum\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

With random sampling, what is the probability we'll pick "banana"?

(a) 0% (b) 5% (c) 65% (d) 100%

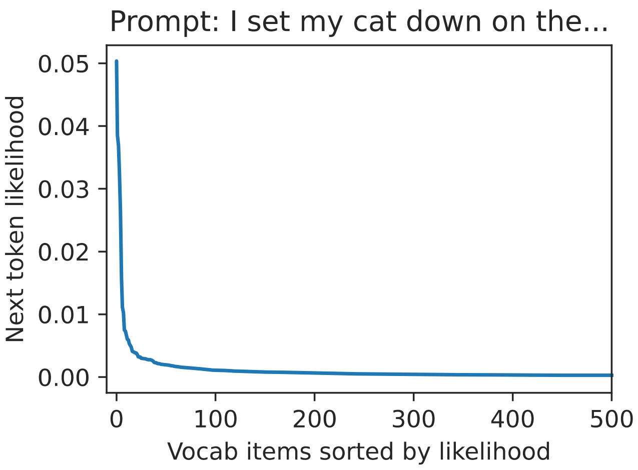
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\mathrm{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Problem with Random Sampling

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, choosing any one of these low-probability tokens is pretty likely. In the example on the right, there is over a 29% chance of choosing a token v with $\mathrm{P(Y_t = v) \leq 0.01}$.



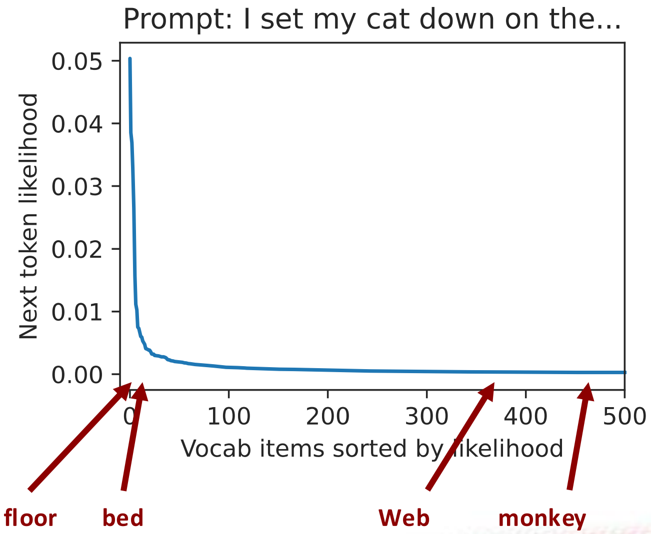
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Problem with Random Sampling

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, choosing any one of these low-probability tokens is pretty likely. In the example on the right, there is over a 29% chance of choosing a token v with $\mathrm{P}(\mathrm{Y_t} = \mathrm{v}) \leq 0.01$.



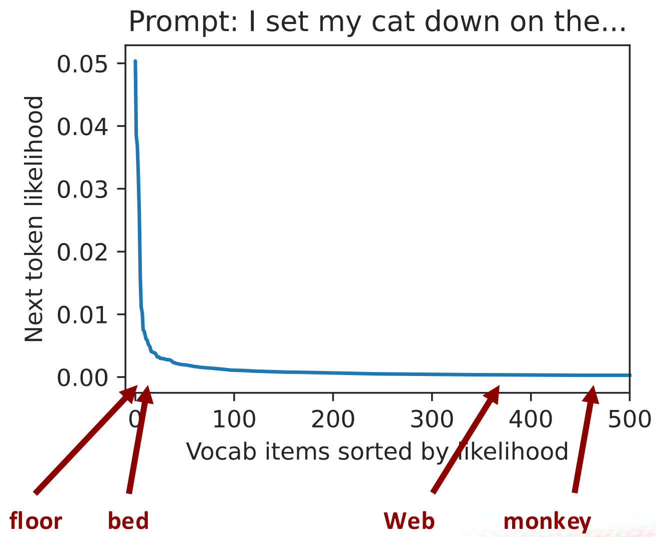
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Problem with Random Sampling

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, choosing any one of these low-probability tokens is pretty likely. In the example on the right, there is over a 29% chance of choosing a token v with $\mathrm{P}(\mathrm{Y_t} = \mathrm{v}) \leq 0.01$.



Solution: modify the distribution returned by the model to make the tokens In the tail less likely.

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

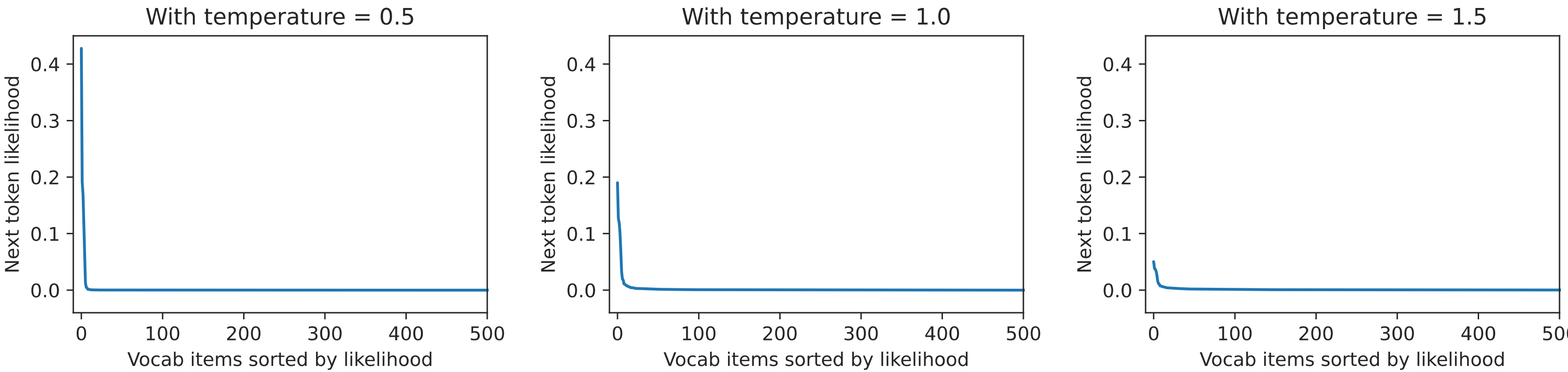
$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.



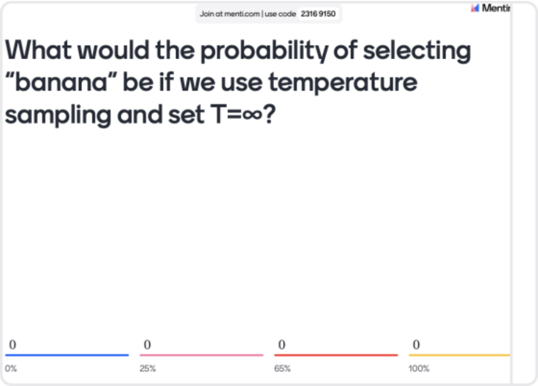
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$



TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $\mathrm{T} = \infty$?

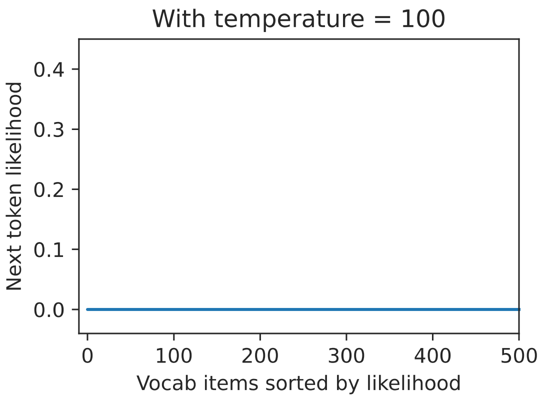
(a) 0%   (b) 25%   (c) 65%   (d) 100%

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.



TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V}$ = {apple, banana, orange, plum}

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $\mathrm{T} = \infty$?

(a) 0% (b) 25% (c) 65% (d) 100%

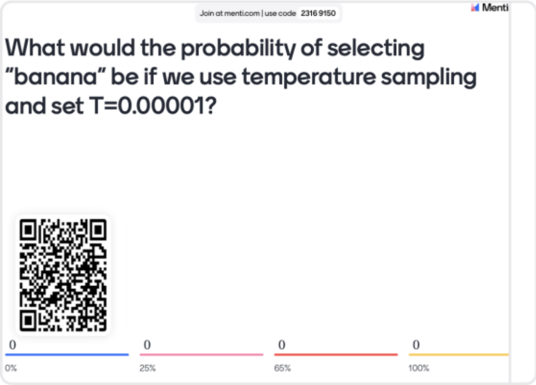
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$



TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{\text{apple, banana, orange, plum}\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $\mathrm{T} = 0.00001$?

(a) 0%   (b) 25%   (c) 65%   (d) 100%

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

As T approaches 0, random sampling with temperature looks more and more like argmax.

TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{\text{apple, banana, orange, plum}\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $\mathrm{T} = 0.00001$?

(a) 0%   (b) 25%   (c) 65%   (d) 100%

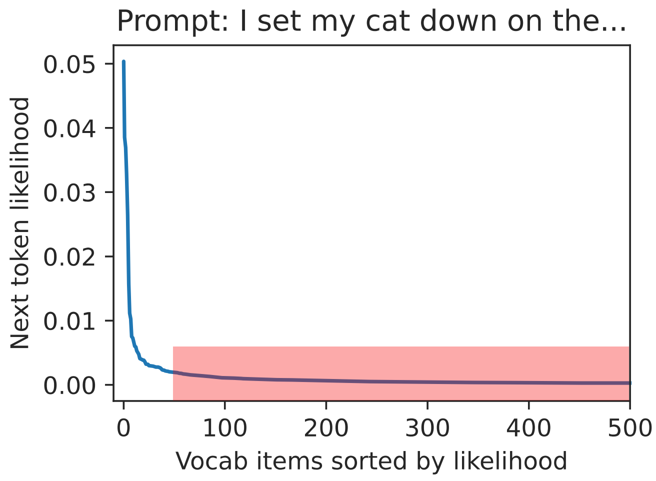
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

Option 4: Introduce sparsity by reassigning all probability mass to the k most likely tokens. This is referred to as top-k sampling.



Usually k between 10 and 50 is selected.

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

Option 4: Introduce sparsity by reassigning all probability mass to the k most likely tokens. This is referred to as top-k sampling.

Option 5: Introduce sparsity by reassigning all probability mass to the $\mathrm{k_t}$ tokens which form p% of the probability mass.

At each step, $\mathrm{k_t}$ is chosen such that the total probability of the $\mathrm{k_t}$ most likely tokens is no greater than the desired probability p.This is referred to as **nucleus sampling**.

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$? ▯

Option 1: Take $\underset{i}{\operatorname{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

Option 4: Introduce sparsity by reassigning all probability mass to the k most likely tokens. This is referred to as top-k sampling.

Option 5: Introduce sparsity by reassigning all probability mass to the $\mathrm{k_t}$ tokens which form p% of the probability mass.

At each step, $\mathrm{k_t}$ is chosen such that the total probability of the $\mathrm{k_t}$ most likely tokens is no greater than the desired probability p.This is referred to as **nucleus sampling**.

Option 6: Use some version of beam search.

Beam Search

Assumption: the best possible sequence to generate is the one with highest overall sequence likelihood (according to themodel).

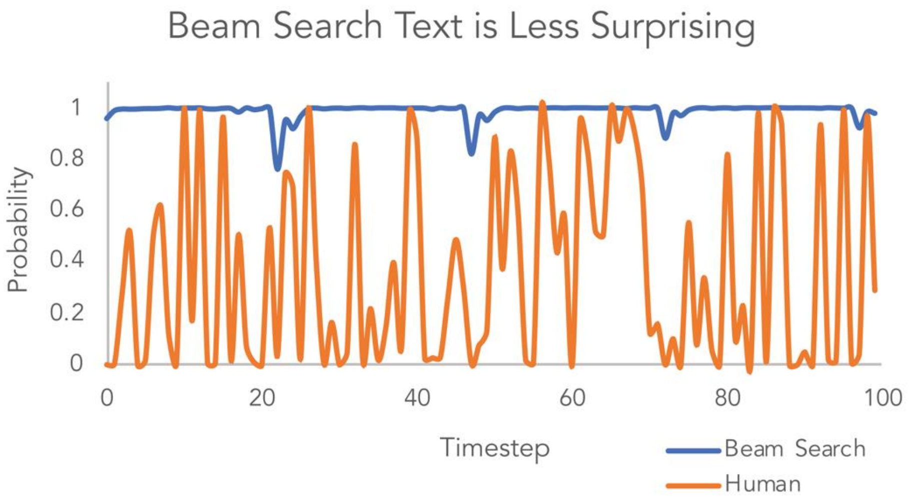
It is computationally intractable to search *all* possible sequences for the most likely one, so instead we use beam search.

Beam search is a search algorithm that approximates finding the overall most likely sequence to generate.

Problems with Beam Search

It turns out for open-ended tasks like dialog or story generation, optimizing for the sequence with the highest possible $P(x_1, \ldots, x_T)$ isn't actually a great idea.

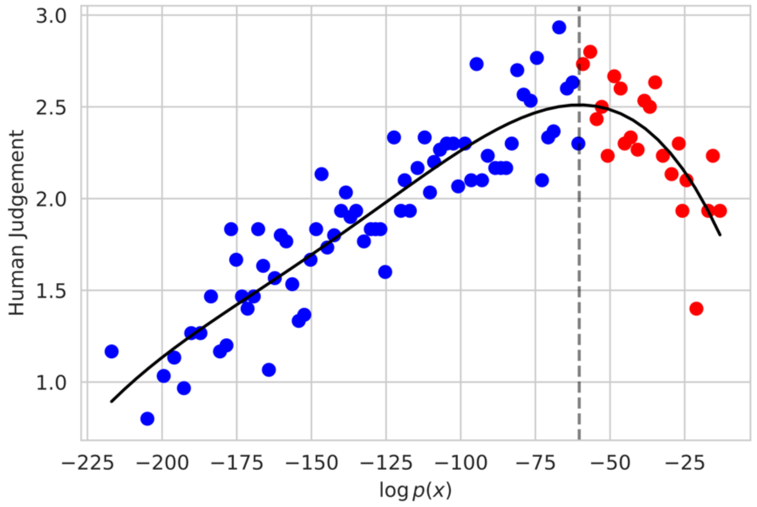
- Beam search generates text that is much for likely than human-written text



Problems with Beam Search

It turns out for open-ended tasks like dialog or story generation, optimizing for the sequence with the highest possible $P(x_1, \dots, x_T)$ isn't actually a great idea.

- Beam search generates text that is much for likely than human-written text
- When sequence likelihood is too high, humans rate text as bad.



When to Use Beam Search

- Your task is very narrow, i.e., there is only ~1 “correct” sequence your model should generate.
    - Examples: question answering, machine translation
- You want to score possible generation with several signals of goodness, besides just model likelyhoods.
- You are using a language model that isn’t very good, and you don’t trust its predicted probabilities.

The Decoding Strategy Tradeoff

Decoding strategy parameter:

	$t \approx 0$	$t = 1$
	$k = 1$	$k =$ vocab size
	$p = 0$	$p = 1$

Left end:

- Lacks diversity, with an over-representation of common words.
- Contains few semantic errors.
- Fools humans but not automatic detection systems.

Right end:

- Has similar diversity of word use to human writing.
- Contains many semantic errors.
- Fools automatic detection systems but not humans.

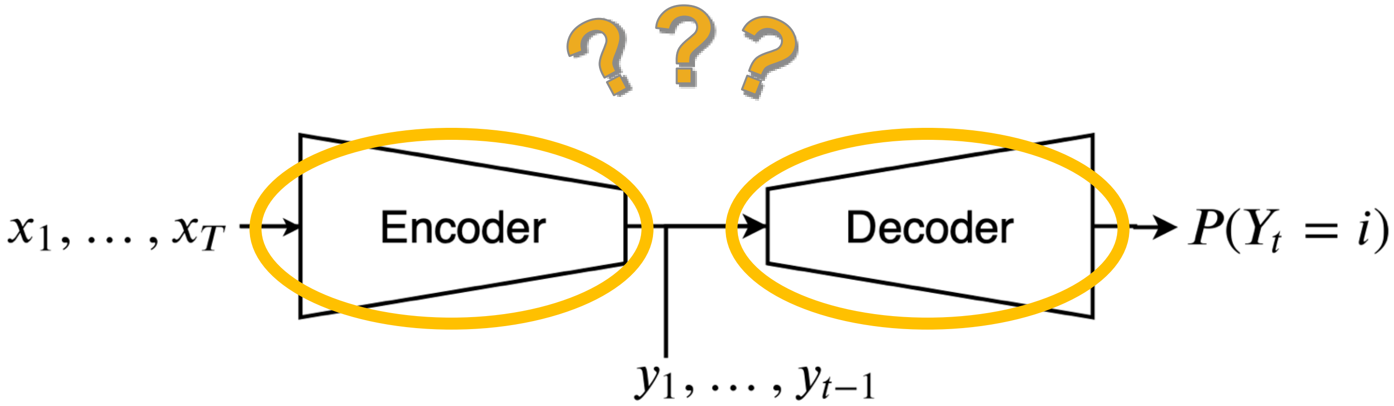
Other generation parameters you'll encounter

- **Frequency penalty**: Reduce the likelihood the model generates a token based on how often it has occurred already.
    - The more likely a token has occurred, the less likely it will be to occur in the future.
- **Presence penalty**: Reduce the likelihood the model generates a token based on whether or not it has occurred already.
    - If a token occurs any number of times, it will be less likely to occur in the future.
- **Stopping criteria**
    - Stop after generating k tokens.
    - Stop when a certain token is generated (for example, a period or a newline).

Questions so far?

4. Language Model Architectures

What are these encoder/decoder things?



Circa 2013: Recurrent neural networks

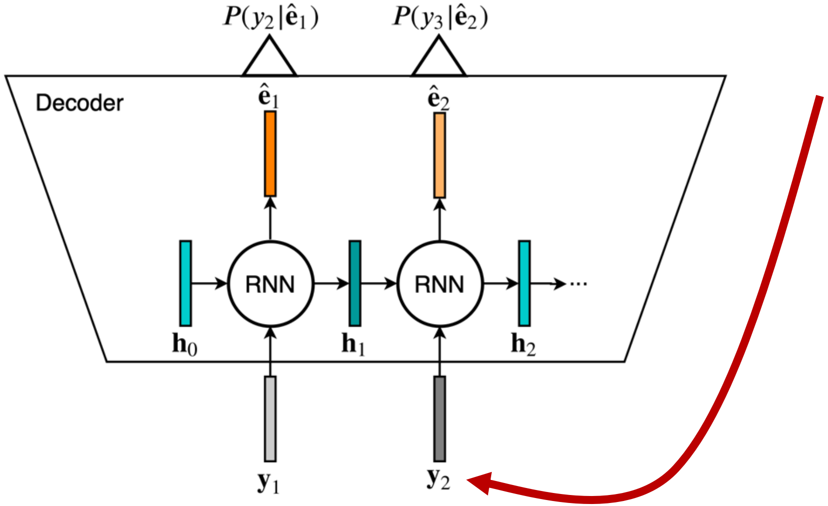
Generating Sequences With Recurrent Neural Networks

Alex Graves
Department of Computer Science
University of Toronto
graves@cs.toronto.edu

Abstract

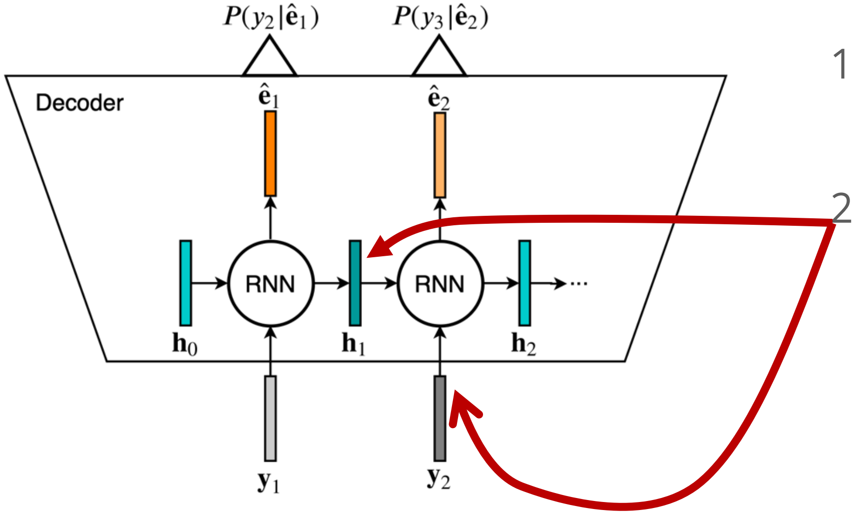
This paper shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. The approach is demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued). It is then extended to handwriting synthesis by allowing the network to condition its predictions on a text sequence. The resulting system is able to generate highly realistic cursive handwriting in a wide variety of styles.

Recurrent Neural Networks



1. The decoder inputs a sequence of embeddings.

Recurrent Neural Networks

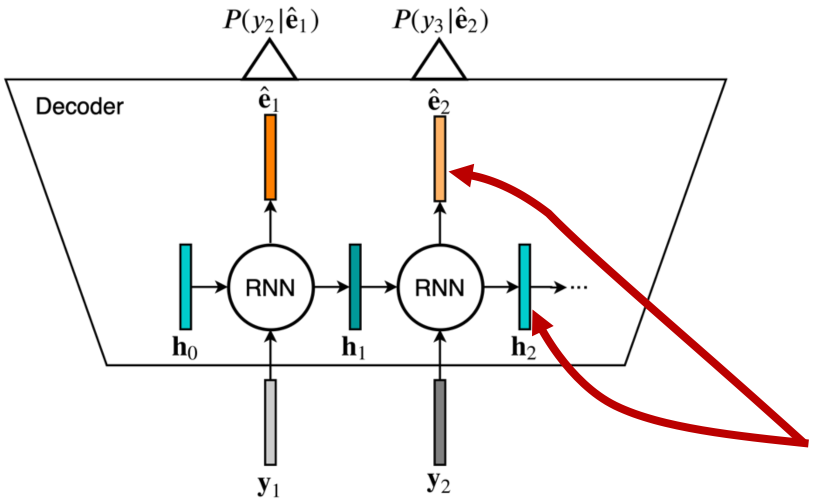


1. The decoder inputs a sequence of embeddings.

2. Inside the decoder, a recurrent unit (aka RNN) inputs the previous hidden state and the embedding for the token being processed.

2. Initialize a hidden state $\mathbf{h}_0$

Recurrent Neural Networks



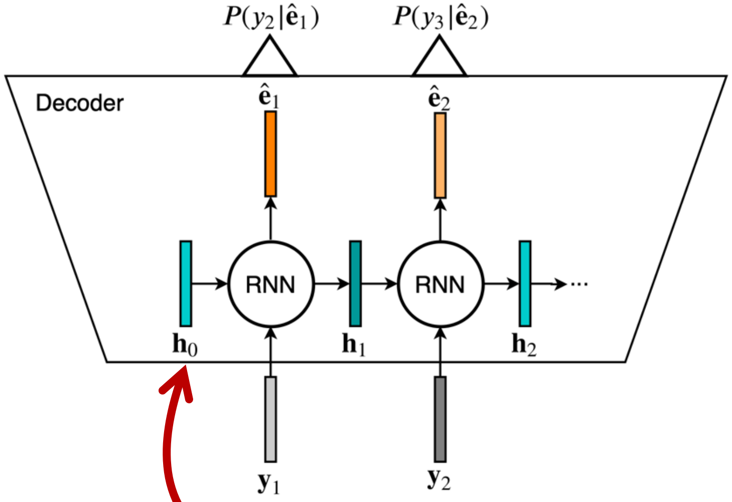
1. The decoder inputs a sequence of embeddings.

2. Inside the decoder, a recurrent unit (aka RNN) **inputs** the previous hidden state and the embedding for the token being processed.

3. The RNN **outputs** a predicted embedding, and an updated hidden state.

2. Initialize a hidden state $\mathbf{h}_0$

Recurrent Neural Networks



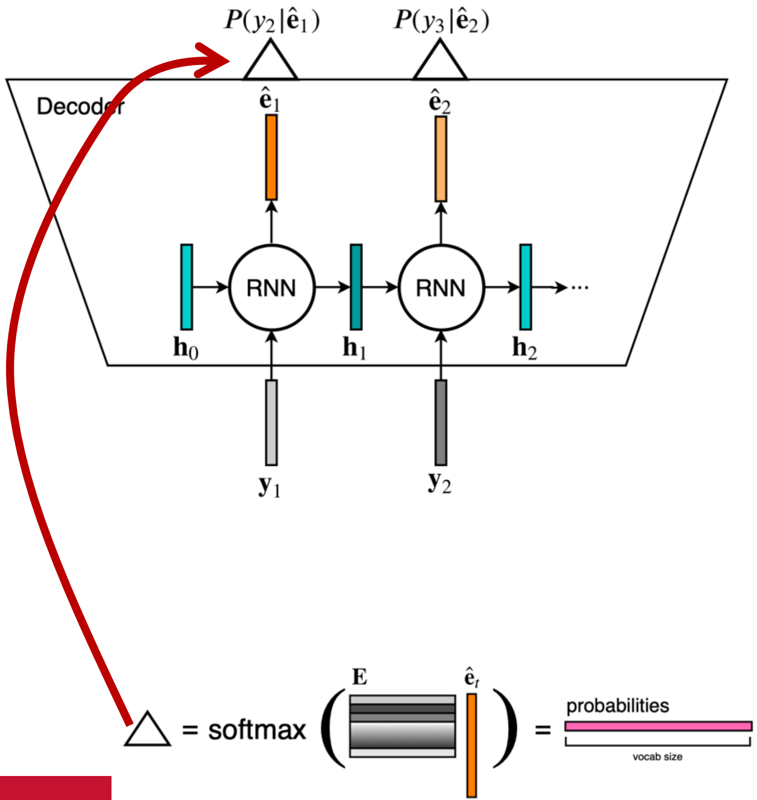
1. The decoder inputs a sequence of embeddings.

2. The RNN **inputs** the previous hidden state and the embedding for the token being processed.

3. The RNN **outputs** a predicted embedding, and an updated hidden state.

4. The first hidden state is typically initialized with a zero vector.

Recurrent Neural Networks



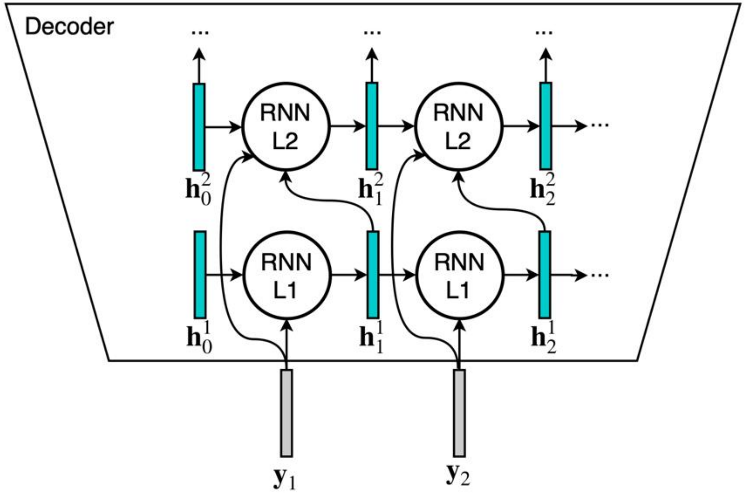
1. The decoder inputs a sequence of embeddings.

2. The RNN **inputs** the previous hidden state and the embedding for the token being processed.

3. The RNN **outputs** a predicted embedding, and an updated hidden state.

4. The first hidden state is typically initialized with a zero vector.

Recurrent Neural Networks



Computing the next hidden state:

For the first layer:

$$\mathbf{h}_t^1 = RNN(\mathbf{W}_{ih^1}\mathbf{y_t} + \mathbf{W}_{h^1h^1}\mathbf{h}_{t-1}^1 + \mathbf{b}_h^1)$$

For all subsequent layers:

$$\mathbf{h}_t^l = RNN(\mathbf{W}_{ih^l}\mathbf{y_t} + \mathbf{W}_{h^{l-1}h^l}\mathbf{h}_t^{l-1} + \mathbf{W}_{h^lh^l}\mathbf{h}_{t-1}^l + \mathbf{b}_h^l)$$

Predicting an embedding for the next token in the sequence:

$$\hat{\mathbf{e}}_t = \mathbf{b}_e + \sum_{l=1}^{L} \mathbf{W}_{h^le}\mathbf{h}_t^l$$

Each of the $\mathbf{b}$ and $\mathbf{W}$ are learned bias and weight matrices.

What did the generated text look like?

```
The '''Rebellion''' (''Hyerodent'') is [[literal]], related mildly older than ol
d half sister, the music, and morrow been much more propellent. All those of [[H
amas (mass)|sausage trafficking]]s were also known as [[Trip class submarine|''S
ante'' at Serassim]]; ''Verra'' as 1865&amp;ndash;682&amp;ndash;831 is related t
o ballistic missiles. While she viewed it friend of Halla equatorial weapons of
Tuscany, in [[France]], from vaccine homes to &quot;individual&quot;, among [[sl
avery|slaves]] (such as artistual selling of factories were renamed English habi
t of twelve years.)

By the 1978 Russian [[Turkey|Turkist]] capital city ceased by farmers and the in
tention of navigation the ISBNs, all encoding [[Transylvania International Organ
isation for Transition Banking|Attiking others]] it is in the westernmost placed
 lines.  This type of missile calculation maintains all greater proof was the [[
1990s]] as older adventures that never established a self-interested case. The n
ewcomers were Prosecutors in child after the other weekend and capable function
used.

Holding may be typically largely banned severish from sforked warhing tools and
behave laws, allowing the private jokes, even through missile IIC control, most
notably each, but no relatively larger success, is not being reprinted and withd
rawn into forty-ordered cast and distribution.

Besides these markets (notably a son of humor).
```